Energy Proportional Computing with OpenCL on a FPGA-Based Overlay Architecture

Awais Hussain Sani, Jose Luis Nunez-Yanez

Department of Electrical and Electronic Engineering, University of Bristol, UK.
Email: {a.sani, j.l.nunez-yanez}@bristol.ac.uk

Abstract— This paper proposes an architecture inspired by ARM big.LITTLE that combines a hardened host with a cluster of soft processors of different complexities, performance and energy profiles. This coarse-grained FPGA overlay architecture results in a hardware accelerator that offers software like programmability, fast compilation, improved design productivity and application portability. A programming flow based on OpenCL is introduced to allow application programmers to implement parallel algorithms at higher level of abstractions. Current OpenCL tools for FPGAs suffer from long compilation times and limited compiler support. Minor changes to the algorithm normally mean full implementation cycles that can take several hours to complete. The proposed architecture allows changes to the application at run-time with cross-compilation done in the host during program execution. To compensate for the loss of performance compared with custom logic the FPGA cluster supports adaptive voltage scaling that enables higher clock frequencies and better adaptation to the program load. Experimental results demonstrates 70% improvement in computational time and 80% reduction in energy consumption by computing OpenCL kernel on different clusters and various operating voltages and frequencies.

I. INTRODUCTION

With Moore's Law approaching its limit, designers and researchers are challenged to increase throughput and reduce energy for the next generation of applications. FPGA accelerators have shown promising advantages in a wide range of applications thanks to increasing logic density and embedded high performance DSP blocks. FPGA accelerators outperform general purpose CPUs in terms of throughput, latency and energy in many applications. However, the expertise required to design hardware, low productivity and long compilation times have generally prevented the utilization of FPGA accelerators in general purpose computing. Different High Level Synthesis (HLS) tools and design languages have been developed to improve design productivity and allow designers to focus on high level functionality. However, designing high performance accelerators still requires low-level implementation expertise that is difficult to acquire for a software programmer. Moreover, it is also difficult and time consuming to maintain an application as an accelerator whose code changes over time.

OpenCL has been designed to improve design productivity. It offers the possibility of transforming FPGA devices into an additional computing resource for software programmers without knowledge of low-level hardware details. However, "OpenCL to FPGA" does not offer a perfect solution to map software into hardware. Performance portability of OpenCL code from the CPU/GPU domain to FPGA is very limited and the accelerator code must be optimized for particular kernel(s). In addition, deployment of this code in the FPGA is costly in terms of implementation time and the need to use low level debugging techniques. To improve the design productivity and make FPGA more accessible to application developers, there is a growing need to propose new solutions and tools to improve the development cycle. Overlay architectures are a promising solution to use OpenCL on a FPGA device that have received significant attention recently.

The contribution of this work can be summarised as follows:

- The design of an architecture overlay with hardened host and a cluster of soft processors that can operate under varying levels of voltage, frequency and logic complexity to adapt performance and energy to the kernel workload demand.
- The construction of a source to source code generator that uses the LLVM Compiler infrastructure to generate optimized C code for the respective OpenCL kernel.
- The partitioning and compilation of the C Code to a set of processors within a cluster in order to reduce the overall system energy consumption by adapting to the application throughput requirement.

The rest of the paper is organized as follows: Section II presents work related to overlay architectures and source-to-source compilers. Section III describes our overlay architecture consisting of clusters of soft processor and energy proportional system used to run OpenCL kernels. The flow of the tool used to generate source-to-source code from OpenCL to C and to generate binaries that run on soft processor is presented in Section IV. Section V describes the benchmarks used in this paper and presents the results obtained after mapping these benchmarks on our cluster of processor using the energy proportional computing system. Finally, we conclude our paper in Section VI with some directions to investigate the overlay architecture in the future.

II. RELATED WORK

To improve the design productivity and to reduce the compilation times, overlay architectures provide an attractive solution for computing kernels on FPGAs. Different types of overlay architectures are developed in the literature such as coarse-grained reconfigurable arrays (CGRAs), overlay based on DSP Blocks Function Units (FUs) and those that used programmable soft cores. CGRAs overlays are designed for

word level computation and on-the-fly customization. For this purpose, a coarse-grained FPGA architecture coupled with an array of general purpose processor has been designed in [1] to obtain software-like programmability and application portability with fast compilation times. An intermediate Fabric (IF) consisting of Computational Units (CUs) has been introduced in [2] to improve compilation time and application portability. The compilation of application kernel is improved with the addition of processing elements (PEs) and crossbar (CB) in IF and through fast memory access and pipelined overlay Functional Units (FUs) in [3]. Overlay architectures have been managed at run-time for configuration and data communication either through an operating system (Linux) [1] or using a hypervisor. These architectures tend to use general purpose PEs that enable application programmers to work at higher levels of abstraction to increase design productivity. However, these architectures have been designed without serious consideration to the underlying FPGA architecture and this results in performance and cost overheads for many applications [4]. This will prevent the practical use of these architectures in FPGA-based systems [3].

To improve throughput and reduce overhead, DSP based overlay architectures using flexible interconnection network have been designed in [5][6]. These architectures use highly pipelined DSP blocks connected through flexible island-style interconnection to map applications on the FPGA. To exploit kernel level parallelism, multiple instances of the kernel are mapped on the underlying overlay architecture. The mapping tool flow consists of C to Data Flow Graph (DFG) transformation, DSP aware mapping, Placement and Routing of FU netlist using VPR (Versatile Place and Route) and latency balancing. However, DSP overlays require low-level FPGA knowledge to fine-tune the mapping in order to improve throughput for different applications. Moreover, designing fully flexible island-style interconnect results in a significant area overhead. To reduce area overhead, linear interconnect is used between pipelined DSP Blocks in DeCO [7]. But still, fine-tuning of mapping tool based on compute kernel is required to get high throughput on different FPGAs. An overlay architecture based on soft processors called IPPro for image processing applications is demonstrated in [8]. The approach has used CAL dataflow language and respective ORC synthesis flow to map application on IPPro to get higher throughput. However, this results in an overlay restricted to particular application that cannot be used for general purpose computing.

One of the problem while porting OpenCL to a cluster of coarse-grained processors is the unavailability of the compiler that generates binaries for that processor starting with the OpenCL code. An alternative is to use source-to-source transformation to convert the OpenCL to C/C++ that can be compiled with the standard processor flow. INSIEME [9] is a source-to-source compiler for C/C++ designed to generate parallel codes (OpenMP, MPI and OpenCL) for heterogeneous multi-core architecture whereas POCL [10] [12] uses Clang and LLVM to produce LLVM *Intermediate Representation* after parsing OpenCL C kernels. In this work we also use this approach to avoid having to write a compiler from scratch.

III. HARDWARE ARCHITECTURE

Our processor-based acceleration cluster is based on a general purpose FPGA-efficient soft processor that supports a standard Instruction set Architecture (ISA), software tools and libraries. The MicroBlaze is selected since it is optimized for implementation in Xilinx FPGAs which are the focus of this work. It is modern, reconfigurable and supports a wide set of applications and has a stable compiler infrastructure consisting of assembler, debugger, simulators and libraries. It supports different embedded operating systems, interface IPs and can work as a standalone accelerator.

ARM big.LITTLE technology has been designed to satisfy the two conflicting requirements of a design: energy-efficient computing and powerful processing. It assigns task to processor cores of different complexity depending on workload demands to reduce energy-consumption. To carry on this concept, an overlay architecture consisting of clusters of MicroBlaze (MB) with different logic complexity and energy proportional system has been designed as shown in Fig. 1.

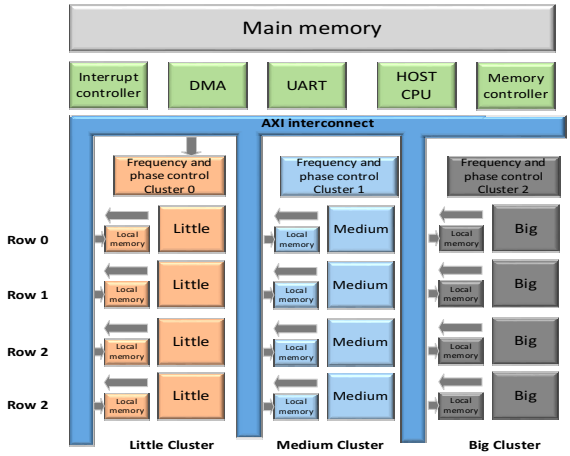

Fig. 1. Overlay Architecture

The clusters are called Little Cluster, Medium Cluster and Big Cluster with regard to the computation capabilities of the processors used in each cluster. Little cluster consists of 4 MB processors configured with 3-stage pipeline and no support for hardware multiplier and Floating-Point Unit (FPU). Medium cluster contains 4 MB processors designed with 5-stage pipeline, hardware support for multiplier and no support for FPU. Big cluster has 4 MB processors each configured with 5-stage pipeline and hardware support for both multiplier and FPU. Computational capabilities and hardware utilization of each processor in a cluster are summarized in Table 1.

Table 1: Hardware utilization of each processor

Cluster	5-stage pipeline	Hardware multiplier/divider/ comparator/shifter	Hardware FPU	Hardware complexity			
				Registers	LUTs	DSPs	BRAMs
Little	NO	NO	NO	361	646	0	8
Medium	YES	YES	NO	1098	1375	3	8
Big	YES	YES	YES	1870	2600	5	8

The energy proportional system consists of in-situ detectors and additional hardware and software to support adaptive voltage and frequency scaling (AVFS). The in-situ detectors [11] are used to protect critical paths in order to reliably operate the design across a range of frequencies and voltages

whereas AVFS is used together with system characterization to influence the voltage and frequency on the fly in closed-loop configuration.

The in-situ detectors have been introduced in the design netlist using the post place & route timing information of the system. The core of the flow is the Elongate tool [11] that transforms the original design netlist into a new netlist with identical functionality and additional power management IP and in-situ detectors. Fig. 2 shows the overall flow that can be decomposed into three distinct phases indicated with numbers 1, 2 and 3 in Fig. 2. During the first phase, the original netlist goes through a full implementation run to obtain post place&route timing data in the form of a text file. In the second stage the Elongate tool takes as input the obtained timing data, the original netlist and Elongate component library that describes the power management core and in-situ detectors and produces the new power adaptive netlist. The third stage consists of a final implementation run of the power adaptive netlist to obtain the device bitstream ready to be downloaded in the device. This third stage is done using the incremental place&route available in the Vivado flow to minimize changes to the implementation caused by the addition of the in-situ detectors and additional logic. The incremental flow enables the reutilization of approximately 98% of the available place&route information.

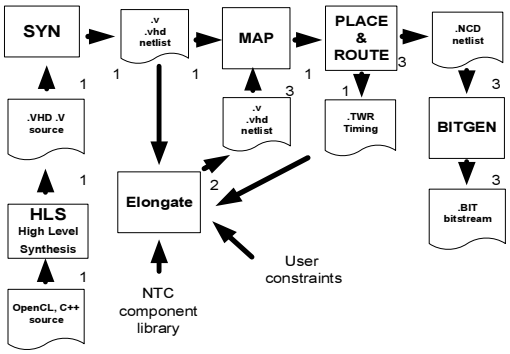

Fig. 2. Implementation Flow of Energy Proportional System

The overall architecture is shown in Fig. 3 that shows 1) the PL voltage domain with the region dedicated to the user IP and the DFS (dynamic frequency scaler) part of the AVFS system 2) the PS voltage domain with the Cortex A9 processors, I2C controller, memory controller and additional peripherals. We have used the ZYNQ-based ZC702 Evaluation Board running UBUNTU Linux OS. The DVS (Dynamic Voltage Scaling) unit originally presented in [11] has been removed since the voltage scaling is done by the ARM directly using the I2C controller shown to the right of the figure. The DFS (Dynamic Frequency Scaler) shows the addition of a PLL whose clock input is connected to the clock output of the MMCM (Mixed Mode Clock Manager). The adaptation control state machine controls the MMCM to generate multiple clock frequencies and resets the PLL to allow it to relock at the different clock frequencies. The control state machine generates up to 535 different frequencies between the ranges of 20 MHz and 400 MHz with fine increments. It constantly monitors the status of the user IP block and decides to increment or decrement frequencies depending on this status. The adaptation control logic also monitors device parameters such as temperature to verify that it remains below allowed maximum. The user IP block is the clusters of coarse-grained processors in charge of performing the useful computation that has been embedded with the logic and memory detectors. The detectors which are shown as EFF in the figure communicate through the EFF control with the adaptation control logic.

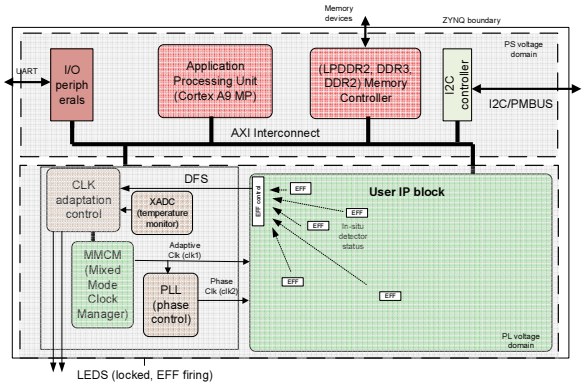

Fig. 3. Overview of the system Design

Fig. 4 shows the detailed description of big cluster with in-situ detectors embedded into the design. The other clusters follow the same design methodology as described in this section. The processors access instructions and data from BlockRAM memories and the critical paths go from the processor logic to the BlockRAM as verified in the timing analysis reports. The detectors are inserted protecting the data and address lines to the BlockRAMs and communicate with the EFF (embedded flip-flop) control visible in Fig. 3. Data and binaries are copied to the BRAM by the ARM host processor through the host AXI directly writing the BlockRAMs or using the hardened PL330 DMA present in the PS side. Once the data and binary copying completes the host processor activates the Microblaze by removing the reset signal. The Microblaze processors proceed to execute the code stored in the BlockRAMs and issue an interrupt to the host once the processing completes. A UART is also available so the processors can report on execution progress. Each processor can execute a different binary or the same binary with different data sets.

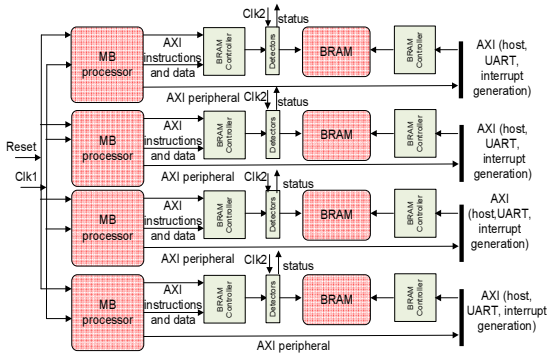

Fig. 4. Microblaze multi-core hardware

IV. TOOL FLOW

The OpenCL program consists of the kernel program and host program. The kernel program defines the single instance of the algorithm that is executed in the index space of the device whereas host program manages and configures the execution of this kernel. In this work, a source to source code translator has been proposed that used Clang frameworks to generate C code for respective OpenCL kernel. The granularity is maintained at the work-group level where multiple work-groups are dispatched to a processor in the cluster. The C code corresponding to each processor is then compiled using MB cross-compiler installed on host processor and loaded for running OpenCL kernel on cluster processors. The host program has been used to support application testing and energy proportional computing on different clusters. The complete Execution flow along with different steps required to generate host and kernel executables are shown in Fig. 5.

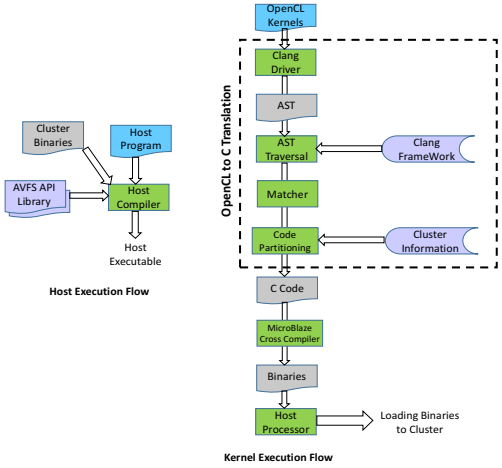

Fig. 5. Host and Kernel Execution Flow

The generation of kernel executable for cluster processors can be divided into two steps: 1) source-to-source translation from OpenCL to C enclosed in dotted lines 2) generation of binaries for each processor in the cluster using MB cross-compiler. Different approaches have been used for source-to-source translation in which one language has been converted into another language at the same abstraction level. One of the most common approach is to generate intermediate representation of the source language, perform suitable transformations and optimizations and output the code for target language. Clang is the collection of compiler frontend API libraries used for parsing, lexing and code analysis. For source-to-source transformation, Clang preprocess the source code by parsing it into Abstract Syntax Tree (AST). The Clang AST Matcher library allows the program to match the nodes or subtree within this AST that follows a specific predicate using different AST traversal techniques. A callback function is registered with each AST matcher and will be called when an AST node matches with its respective AST matcher. This callback function makes it possible for the program to change the source code or/and generate code in target language.

Fig. 6 explains the whole process of transformation from OpenCL to C as performed in this work with a simple example. Fig. 6.a shows the OpenCL kernel program for a vector addition. After parsing this program into AST, the Clang traverse this tree for matching the nodes for which AST matcher and callback functions are registered for translating OpenCL into C. After performing the necessary transformations, the output C code is shown in Fig. 6.b. The code is further partitioned at the work-group level by using the cluster information. Fig. 6.c depicts the code after partitioning that runs on first processor of the cluster.

```
__kernel void vec_add(__global const int *a, __global const int *b, __global int *c, int numElements){
    int gid = get_global_id(0);
    c[gid] = a[gid] + b[gid];
}
```
(a)

```
void vec_add(const int *a, const int *b, int *c, int numElements){
    int index_space;
    for(index_space =  0 ; index_space < numElements ; index_space++) {
    int gid = index_space;
    c[gid] = a[gid] + b[gid];
    }
}
```
(b)

```
void vec_add(const int *a, const int *b, int *c, int numElements){
    int index_space;
    for(index_space =  0 ; index_space < (numElements/4) - 1 ; index_space++) {
    int gid = index_space;
    c[gid] = a[gid] + b[gid];
    }
}
```
(c)

Fig. 6. OpenCL-to-C Transformation for MB cluster

The time required by source to source compiler to transform different OpenCL kernels into C is shown in Table 2.

Table 2: Execution Time: source to source compiler

Kernel	Vector Addition	Vector Multiplication	Matrix Multiplication	Image Convolution
Time in secs	0.3	0.3	0.5	0.7

The cross-compiler is used to build a binary on one platform called host platform that will be run on another platform called target platform. In our case, the host platform is the ARM processor and the target platform is MicroBlaze Processor. To completely execute the Kernel Execution Flow from OpenCL to MB binaries on host platform, both Clang Framework and MB cross-compiler are installed on the ARM Processor. MB cross-compiler is a little ending GNU compiler toolchain that is used to generate MB binaries for AXI based Interconnect system. Compiler also uses flags for activating different processor features and generating instructions that are executed by hardware components present in the processors of different clusters such as hardware multiplier, divider, Floating Point Unit (FPU) and barrel shifter.

The Host Execution Flow is used to load the generated MB binaries onto the processors of a cluster identified by user and to test the execution of OpenCL kernel on it. It is also used to clock gate the processors of the cluster not involved in the computation of the kernel so that the actual performance and energy consumption values can be computed for a particular cluster. The host also executes the software daemon that monitors the status of the in-situ detector and writes control instructions to the DFS (frequency control) and I2C (voltage control) units using the AVFS API library. For these experiments, the daemon records the temperature, frequency, operating voltage, FPGA power and detector status. Running the host and the kernel execution flow on the same platform enables the execution of the OpenCL kernel directly on different clusters under control of the energy proportional system to investigate the energy and performance tradeoffs of the different benchmarks.

V. EXPERIMENTAL RESULTS

To compare the performance and energy consumption of different clusters within the architecture, four OpenCL benchmarks are executed under varying level of frequencies and voltages. These benchmarks are vector addition, vector multiplication, matrix multiplication and image convolution. The energy proportional system is designed to search for an optimal frequency for a given voltage value. In the test system, the valid range of voltages extends from 0.7 to 1V. The experiment starts by loading data and binaries for a particular benchmark to the BRAMs of a cluster specified by the user. To configure the voltage and frequency of the system, the daemon first configures the voltage regulator for a particular voltage and then activates the DFS unit to searches for the highest possible frequency for that voltage. Once DFS unit detects this point using in-situ detectors and reports to the host, the daemon configures the voltage regulator with a different voltage from the range of valid voltages and restarts the process. It is important to note that the benchmarks are running on the selected clusters in parallel continuously to emulate the real world implementation of benchmarks on that cluster for range of frequencies and voltages.

Fig. 7 shows the maximum clock frequencies detected by daemon for a nominal voltage of 1.0 to a low voltage of 0.7V for different clusters using matrix multiplication as a benchmark. The figure depicts that there is a linear relationship between frequency and voltage and more importantly the detector fires for these clusters in the range from 165 MHz to 175 MHz that is much higher than the worst case frequency of 100 MHz reported by tools after timing analysis. This difference encourages the exploration of the performance and power margins that could be exploited depending on the kernel work load.

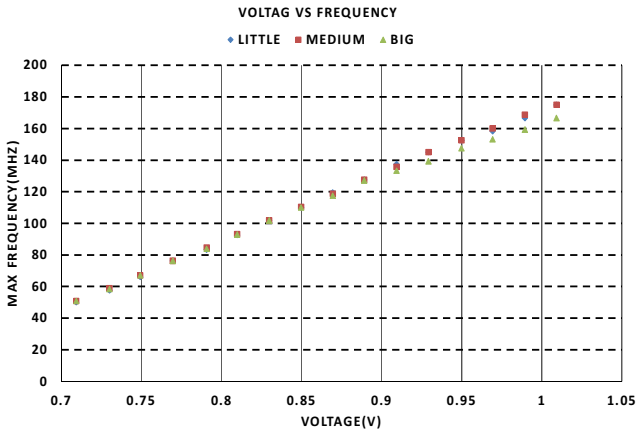

Fig. 7. Voltage and Frequency analysis for different clusters

Different experiments have been performed to explore the possible power and performance gains for different cluster by computing to the limit and detecting the optimal power, execution time and energy consumption levels for various OpenCL benchmarks. For these experiments, vector addition, vector multiplication and image convolution benchmarks are executed using integer data type whereas matrix multiplications is run using floating point data type to measure the efficiency of various clusters. Fig. 8. shows the total power, total execution time and energy consumption values by computing selected OpenCL benchmarks at nominal voltage of 1.0V and minimum operating voltage of 0.7V on the different clusters. The data type used in a particular benchmark is also mentioned in Fig. 8 within small brackets.

From a power consumption perspective, as depicted in Fig. 8.a, it is clear that the Little cluster is more power efficient than the rest of the clusters and there is significant power reductions possible by operating different clusters at 0.70V as compared to operating them at nominal voltage of 1V. It is also clear that cluster with more computational capabilities consume more power as compared to the rest of the clusters mainly due to its higher logical complexity.

However, execution time analysis illustrates that Medium cluster takes less time to run the vector multiplication and image convolution benchmarks than the rest of the clusters as shown in Fig. 8.b. This is mainly due to the presence of hardware multiplier that accelerates the multiplication process for these benchmark. In addition, Big Cluster executes matrix multiplication much faster as compared to the other clusters due to the presence of FPU that speeds up the processing of floating point data present in the benchmark. Moreover, in case of the vector addition benchmark, execution time is almost the same for each cluster. It is also clear that execution time can be improved by almost 70% by operating clusters at 1V as compared to operating them at 0.70V.

From previous analysis, it is evident that reducing the operating voltage/frequency increases the execution time of the benchmark and could result in detrimental overall energy effect since energy is the product of power and execution time. Therefore, care must be taken while selecting operating voltage/frequency and the cluster that results in minimum energy consumption given a throughput requirement of the benchmark. In our analysis of energy utilization, a significant reduction in energy consumption is witnessed by running the vector multiplication and image convolution benchmarks on Medium cluster mainly due to the acceleration of multiply function and by executing the matrix multiplication on Big Cluster because of the presence of FPU. However, little cluster is more energy efficient for running the vector addition benchmark since it consumes less power than the rest of the clusters as illustrated in Fig. 8.c. It is also clear that energy consumption is almost 80% lesser at lower voltage of 0.7V as compared to the nominal voltage of 1V.

The experiments show that the energy proportional system and clusters of different computational capabilities enable the creation of adaptable system. This system allows OpenCL kernel to be run at lower energy consumption levels or at higher performance level or on specific cluster depending on operating conditions and workload size requirements.

VI. CONCLUSION

In this paper, an overlay architecture consisting of clusters of soft processor each having different logical and computational complexity along with an energy proportional system based on

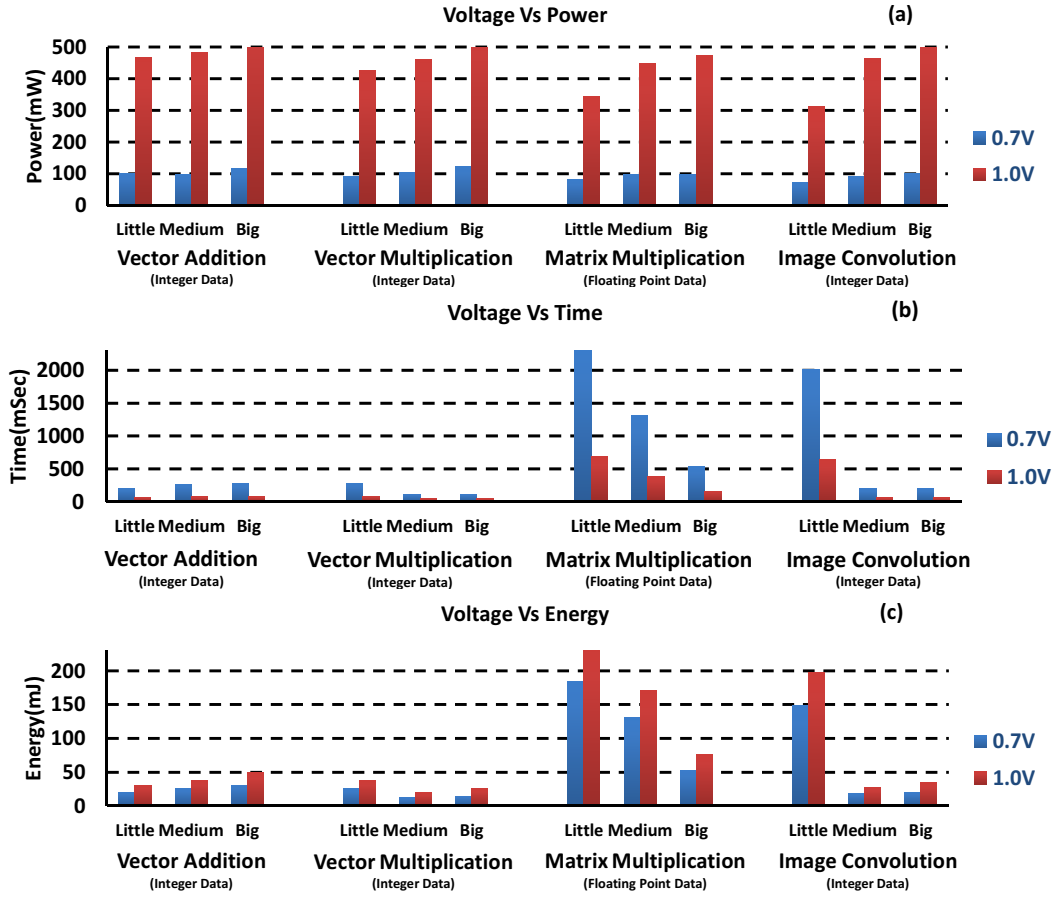

Fig. 8. (a) Power (b) Time (c) Energy analysis for different Benchmarks on various Clusters

frequency and voltage scaling has been designed. A source to source compiler has been proposed to run OpenCL kernel directly as a software on our architecture. Initial experimental result shows that our architecture provides the ability of selecting the operating voltage/frequency and a cluster for a particular benchmark in accordance with its kernel workload, performance and energy consumption requirements. Future work involves exploring more complex benchmarks and developing a scheduling algorithm that will enable the daemon running on the host processor to automatically assign points of voltage and frequency to the overlay architecture according to performance and energy consumption requirements of the benchmarks.

ACKNOWLEDGEMENT

This work is supported by the UK EPSRC under grants ENPOWER (EP/L00321X/1) and ENEAC (EP/N002539/1) grants.

REFERENCES

[1] J. Cong, H. Huang, C. Ma, B. Xiao, and P. Zhou, "A fully pipelined and dynamically composable architecture of CGRA," *IEEE Symposium on FPGAs for Custom Computing Machines (FCCM)*, 2014.

[2] J. Coole and G. Stitt, "Intermediate fabrics: Virtual architectures for circuit portability and fast placement and routing," *Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, 2010, pp. 13–22.

[3] J. Benson, R. Cofell, C. Frericks, C.-H. Ho, V. Govindaraju, T.Nowatzki and K. Sankaralingam, "Design, integration and implementation of the DySER hardware accelerator into OpenSPARC," *International Symposium on High Performance Computer Architecture (HPCA)*, 2012.

[4] A. K. Jain, X. Li, S. A. Fahmy, and D. L. Maskell, "Adapting the DySER architecture with DSP blocks as an Overlay for the Xilinx Zynq," *International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART)*, 2015.

[5] A. K. Jain, S. A. Fahmy, and D. L. Maskell, "Efficient Overlay architecture based on DSP blocks," *IEEE Symposium on FPGAs for Custom Computing Machines (FCCM)*, 2015.

[6] A. K. Jain, D. L. Maskell, and S. A. Fahmy, "Throughput oriented FPGA overlays using DSP blocks," *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, 2016, pp. 1628–1633.

[7] Abhishek Kumar Jain, Xiangwei Li, Pranjul Singhai, Douglas L. Maskell and Suhaib A. Fahm, "DeCO: A DSP Block Based FPGA Accelerator Overlay With Low Overhead Interconnect", *International Symposium on Field-Programmable Custom Computing Machines*, 2016.

[8] F. M. Siddiqui, M. Russell, B. Bardak, R. Woods, and K. Rafferty, "IPPro: FPGA based Image Processing Processor," *IEEE Workshop on Signal Processing Systems (SiPS)*, Oct 2014.

[9] H. Jordan, P. Thoman, J. Durillo, S. Pellegrini, P. Gschwandtner, T. Fahringer, H. Moritsch. "A Multi-Objective Auto-Tuning Framework for Parallel Codes", *Interntional Conference for High Performance Computing, Networking, Storage and Analysis (SC 2012)*, Nov. 2012,

[10] Pekka Jääskeläinen et al., "pocl: A performance-portable OpenCL implementation" *International Journal of Parallel Programming* Page. 1–34 (2014)

[11] Nunez-Yanez, J.L., "Adaptive Voltage Scaling with In-Situ Detectors in Commercial FPGAs," *Computers, IEEE Transactions on* , vol.64, no.1, pp.45,53, Jan. 1 2015

[12] Mohammad Hosseinabady, Jose Luis Nunez-Yanez., "Optimised OpenCL workgroup synthesis for hybrid ARM-FPGA devices," *FPL 2015.*